Brian Yoo (bgy2 | 140007707)
Jeffrey Kang (jk976 | 13900087)
Jarrett Mead (jfm168 | 143008288)
CS 352: Internet Technology
Phase 3

Github: https://github.com/bgyoo970/Java-BitTorrent-Client

As before in phase 2, we are able to correctly interface with the tracker and give it periodic updates based on the min_interval value given by the meta info from the torrent file. It then creates a handshake and verifies it against the peers the client tries to download from. Depending on the state of the download (peer) thread, it will update the tracker with a value of "started", "completed", or "stopped". The download thread will then receive bitfield messages, which will check what pieces that the peer has and determine rarity based on the bitfield message. The rarest piece is determined by seeing what pieces the peer has from the bitfield message, and storing it into a hash of an initial quantity of one. If another peer has the same piece, then it will store it into the hash, see a collision in the index of one, and store the entry into an index of two. This will then represent that there are two of the same piece among the peers. The higher the index, the less rare the piece is.

After determining the rarity of each piece from the list of peers, the download threads will then begin to download each piece based on the rarest piece first. If multiple pieces have the same rank in rarity, then a random number generator will help determine a randomized selection on which piece to download next.

For upload, a single listen thread will be looking for any incoming connections. If the handshake is correctly verified, then an upload thread will be created. This way, multiple upload threads can be created and run in parallel simultaneously as the listen thread will always be listening for incoming connections. Have messages will be sent to the peer, the peer will then send an interest message if it wants to download pieces. An un-choke message will then be sent and the upload thread can then initiate uploading to the given connection.

A separate Gui class and GuiThread class was made to help create the user interface for the BitTorrent client. A separate GUI thread is responsible for starting and stopping the main. This interface will keep track of the torrent name, file name, port number, amount downloaded, amount remaining, amount uploaded, and a progress bar to keep track of the percent completed.

Brief description of classes:
RUBTClient: Initializes objects and starts all necessary threads to upload and download from peers. Torrent: Single object that holds most of the metadata from the torrent file.
ThreadHandler: Contains methods for the main to run. It helps set up IO streams and start/run all of the threads created in main.

Peer: Responsible for the methods of creating/verifying handshakes, opening TCP connections to respective peers.
PeerThread: Responsible for creating download threads per peer, sends interest messages and downloads each piece.
ListenThread: Creates a listening thread. It first creates a socket connection and listens for any incoming connection from peers. Then it tries to verify handshakes with any incoming connections and starts an
upload thread to start seeding with any valid peers.
UploadThread: Takes in a message and checks if it is an interested message. If so, then an un-choke message is reciprocated and the thread then waits for requests messages. Each request message will then be serviced as a piece message is created and send back to the seed. Message: Contains methods to send interest messages, commit to downloading a file, and generating peer messages based on length prefix, message id, and payload.
Download: Object that holds information for the tracker (i.e. downloaded, left, event) and a synchronized methods to help the threads download simultaneously.
UpdateTrackerThread: Responsible for creating a thread that will run during the minimum interval value given from the torrent file. This thread will update the tracker for each passing interval of time.
UserInputThread: Responsible for creating a thread that will talk to all other threads and tell them to stop running if the user prompts to quit from the program.

Labor Division:
Brian and Jarrett were responsible for the rarest-piece-first selection algorithm for this phase along with maintaining multiple connections with download threads and upload threads. Jeff was responsible for creating the GUI and integrating it into the main code.